(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
23 January 2003 (23.01.2003)

PCT

(10) International Publication Number
WO 03/007147 A1

(51) International Patent Classification[7]: G06F 7/38

(21) International Application Number: PCT/RU01/00286

(22) International Filing Date: 13 July 2001 (13.07.2001)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant *(for all designated States except US)*: **INTEL ZAO** [RU/RU]; Sokol-10 Business Center, Chapaevsky per., 14, Moscow, 125252 (RU).

(71) Applicant *(for US only)*: **NORIN, Robert, Samuel** [US/US]; 14170 SW 119th Place, Tigard, OR 97224 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **NARAIKIN, Andrei Alexeevich** [RU/RU]; ul. Rodionova, 9-41, Nizhny Novgorod, 603093 (RU). **DRYZHAKOVA, Olga Vladimirovna** [RU/RU]; ul. Rodionova, 41-9, Dzerzhinsk, 606034 (RU). **ISAEV, Alexandr Vladimirovich** [RU/RU]; ul. Brinskogo, 17-2/3, Nizhny Novgorod, 603163 (RU).

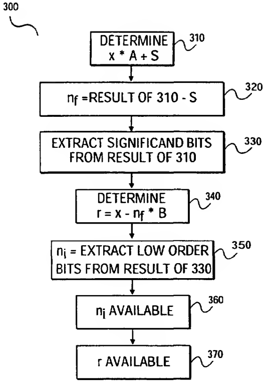
(74) Agent: **TKACHENKO, Valeria Sergeevna**; 000 "Sojuz patent", ul. Iliinka, 5/2, Moscow, 103735 (RU).

(81) Designated States *(national)*: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, HR, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

[Continued on next page]

(54) Title: A METHOD AND APPARATUS TO EXTRACT INTEGER AND FRACTIONAL COMPONENTS FROM FLOATING-POINT DATA



(57) **Abstract:** A method is presented including decomposing a first value into many parts. Decomposing includes shifting (310) a rounded integer portion of the first value to generate a second value. Generating (320) a third value. Extracting (330) a plurality of significand bits from the second value to generate a fourth value. Extracting (340) a portion of bits from the fourth value to generate an integer component. Generating (350) a fifth value. Also the third value, the fifth value, and the integer component are either stored (360, 380) in a memory or transmitted to an arithmetic logical unit (ALU).

WO 03/007147 A1

Published:
— *with international search report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

A METHOD AND APPARATUS TO EXTRACT INTEGER AND FRACTIONAL COMPONENTS FROM FLOATING-POINT DATA

BACKGROUND OF THE INVENTION

Field of the Invention

This invention relates to processing computations, and more particularly to a method and apparatus for reducing floating-point operations necessary to extract integer and fractional components.

Description of the Related Art

In many processing systems today, such as personal computers (PCs), mathematical computations play an important role. Numerical algorithms for computation of many mathematical functions, such as exponential and trigonometric operations, require the decomposition of floating-point numbers into their associated integer and fractional parts. These operations may be used for argument reduction, indexes to table values, or for the construction of a result from a number of constituent elements. Many times, decompositions of floating point numbers into their integer and fractional parts occur in the critical computational path. As a result, the speed at which the mathematical functions may be executed are often times limited.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention is illustrated by way of example and not by way of limitation in the figures of the accompanying drawings in which like references indicate similar elements. It should be noted that references to "an" or "one" embodiment in this disclosure are not necessarily to the same embodiment, and such references mean at least one.

Figure 1 illustrates the ANSI/IEEE standard 754-1985, IEEE standard for binary floating-point arithmetic, IEEE, New York 1985 (IEEE) representation for a single precision floating-point, double precision representation and double extended precision representation.

Figure 2 illustrates a typical method for computing integer and floating point numbers for an equation.

Figure 3 illustrates an embodiment of the invention that reduces the number of floating point operations necessary to compute integer and fractional components.

Figure 4 illustrates an embodiment of the invention used to generalize selection of a constant S.

Figure 5 illustrates a typical process for loading constants and calculating the necessary coefficients for decomposition of floating-point numbers into their integer and fractional parts.

Figure 6A-B illustrates an embodiment of the invention for loading of constants and performing decomposition of floating-point numbers into their integer and fractional parts.

Figure 7 illustrates an embodiment of the invention having a computational component.

DETAILED DESCRIPTION OF THE INVENTION

The invention generally relates to a method to reduce the number of floating point operations necessary to extract integer and fractional components. Referring to the figures, exemplary embodiments of the invention will now be described. The exemplary embodiments are provided to illustrate the invention and should not be construed as limiting the scope of the invention.

Figure 1 illustrates the ANSI/IEEE standard 754-1985, IEEE standard for binary floating-point arithmetic, IEEE, New York 1985 (IEEE) representation for a single precision floating-point representation 105, double precision representation 106, and double extended representation 107. The IEEE single precision representation 105 requires a 32-bit word. This 32-bit word may be represented as bits numbered from left to right (0 to 31). The first

bit, F 110, is a sign bit. The next eight bits, labeled E 120, are exponent bits. The final 23 bits, 9 through 31, represented as F 130, are the fractions (also known as the significand).

For IEEE double precision representation 106, F 110 is a sign bit, E 140 are the exponent bits (11 bits), and the final representative bits, F 150, are the 52 fraction representation bits (also known as the significand).

For IEEE double extended precision representation 107, F110 is a sign bit, E 160 are the exponent bits (15 bits), and the final representative bits, F 170, are the 64 fraction representation bits (also known as the significand).

As an example of the decomposition of floating-point numbers into their integer and fractional parts, the following equations are presented to illustrate one such example:

Given

$w = x*A$ (Equation 1)

where $A = 1/B$ (Equation 2)

Find n and r where $x = n*B + r$ (Equation 3)

where n is a whole number, and A, B, r, w and x are floating-point quantities. Therefore, the problem may be restated as: given an input argument, x, and constants A and B, how many times n does the value B occur in the value x, and what is the remainder? Moreover, n is often used as an index to perform a table lookup, or as the exponent of a subsequent quantity such as 2^n. Therefore, n needs to be represented both as an integer (n_i), and as a floating-point quantity (n_f). Thus, three quantities are needed from the computation: n_i (n as an integer), n_f (n as a floating-point value) and r as a floating-point value.

Figure 2 illustrates a typical method for computing n_i, n_f, and r. In **Figure 2**, process 200 begins with block 210 where $w=x*A$. Block 220 converts w to an unnormalized rounded integer. The value computed in block 220 is then used in block 230 to compute n_f by having this number normalized as a whole number. Block 240 also uses the value from block 220 and then computes n_i by converting the value from block 220 to an integer. In block 250, n_i is available to be transferred to an arithmetic logical unit (ALU) or stored in

memory. In block 260, r is computed by subtracting the quantity of n_f*B from x. In block 270, r is available to be transferred to an ALU or stored in memory.

Table I illustrates the typical method of computing n_i, n_f, and r in terms of instruction level pseudo-code. As can be seen from **Table I**, there are three floating point operations handled by a floating-point arithmetic and logic unit (Falu), and one integer operation handled by an integer arithmetic and logical unit (Ialu). Note that the numbers in parentheses refer to cumulative instruction cycle count (latency) for a processor such as an Intel Itanium™ processor.

TABLE I

Falu op 1:	w=x*A	(1)
Falu op 2:	w_rshifted=convert_to_unnormalized_rounded_int(w)	(6)
Falu op 3:	n_f=convert_to_normalized_whole_number(w_rshifted)	(13)
Ialu op 1:	n_i=convert_to_integer(w_rshifted)	(14)
	n_i available	(18)
Falu op 4:	$r=x-n_f*B$	(18)
	r available	(23)

Figure 3 illustrates an embodiment of the invention that reduces the number of floating point operations necessary to compute n_i, n_f, and r. Process 300 begins with block 310 which computes x*A+S, where S and A are constants and x is a floating-point number. In one embodiment of the invention, the constant S is chosen such that the addition of S to x*A will shift the rounded integer portion of x*A into the rightmost bits of the significand. Block 320 then computes n_f by subtracting S from the value computed in block 310, thus creating an integer value. Block 330 creates n_i+S by extracting the significand bits from the resulting value from block 310. Block 340 computes r by subtracting the quantity of n_f*B from x. Block 350 extracts low ordered bits from the value computed in block 330, resulting in n_i. In block 360, n_i is available to be transmitted to an ALU or stored in memory. In block 370 r is available to be transmitted to an ALU or stored in memory.

Table II illustrates the embodiment of the invention reducing floating-point operations in instruction-level pseudo-code. Note that as an example, the numbers in parentheses refer to cumulative instruction cycle count (latency) for a processor such as an Intel Itanium™ processor. In one embodiment of the invention, the constant S is chosen such that the addition of

S to x*A will shift the rounded integer portion of x*A into the rightmost bit of the significand. Therefore, S can be converted into the integer, n_i, after one Falu operation instead of two. Moreover, the floating-point representation, n_f, can be directly obtained by a second Falu operation that subtracts S from the first Falu result. It can be seen that the desired quantities are obtained with one less Falu instruction. Thus, the embodiment of the invention results in a savings of seven cycles of overall latency on a processor, such as an Intel Itanium™ processor.

Table II

Falu op 1:	w_plus_S_rshifted=x*A+S	(1)
Falu op 2:	n_f=w_plus_S_rshifted-S	(6)
Ialu op 1:	ni_plus_S=extract_significand_bits(w_plus_S_rshifted)	(9)
Falu op 3:	r=x-n_f*B	(11)
Ialu op 2:	n_i=extract_low_order_bits(ni_plus_S)	(11)
	n_i available	(12)
	r available	(16)

A performance benefit also accrues to many software pipeline loops involving this embodiment of the invention. Many loops are resource limited by the number of floating-point instructions required by the computation. Since, this embodiment of the invention involves one less floating-point instruction than a typical method, maximum throughput for the loop is increased.

The following discussion relates to the selection of the constant S in one embodiment of the invention. For ease of discussion, suppose the floating-point representation contains b bits in the significand (e.g., b = 64), an explicit integer bit, and b-1 bits of fraction. The exponent field of the floating-point representation locates the binary point within or beyond the significant digits. Therefore, the integer part of a normalized floating-point number can be obtained in the right-most bits of the significand by an unnormalizing operation, which shifts the significand b-1 bits to the right, rounds the significand, and adds b-1 to the exponent. The significand contains the integer as a b-bit, 2's complement integer. The low-order bits of the significand containing the integer part of original floating-point number can be obtained by adding to the number, a constant 1.10 ..... 000*2^{b-1}. This constant, is one value of S selected in one embodiment of the invention.

The resulting significand contains the integer as a (b-2) bit 2's complement integer. The bit immediately to the left of the b-2 zeros in the fractional part is used to ensure that for negative integers the result does not get renormalized, thereby shifting the integer left from its desired location in the rightmost bit of the significand. If fewer than b-2 bits are used in the subsequent integer operations, then the instructions in **Table II** are equivalent to those of **Table I** for computing n_i, n_f, and r.

In one embodiment of the invention the selection of S can be generalized if the desired result is to be m, where $m=n*2^k$. In this case, the exponent of the constant would be (b-k-1). In this embodiment, the selection of S is useful when the desired integer needs to be divided into sets of indices for a multi-table lookup. For example, n may be broken up such that $n=n_0*2^7 + n_1*2^4 + n_2$ to compute indices n_1 and n_2 for accessing 16-entry and 8-entry tables. With this embodiment, it is required that S be available at the same time as the constant A. In one embodiment of the invention, the constant S can be loaded from memory or on a processor such as Intel's Itanium™, S is easily generated with the following instructions 1) movI of the 64-bit IEEE double precision bit pattern, followed by 2) setf.d to load S into a floating-point register.

In one embodiment of the invention, the constant may be of the form having a "1" followed by a decimal point, j-1 bits ("0"s or "1"s) to the immediate right of the decimal point, a "1" following the j-1 bits, then b-j-1 "0"s. Note that the previous discussed embodiment was of the form having j=1.

The following discussion relates to an embodiment of the invention incorporating the creation of constants needed to compute n_i, n_f, and r. Accuracy requirements of mathematical library algorithms typically require the multiplication, w=x*A, be performed in double-extended precision (64-bit space significand). Therefore, the constant A needs to be loaded with double-extended precision. This is typically performed by storing the constant statically in memory, then loading it into a floating-point register (e.g., the ldfe instruction on an Intel Itanium™ processor).

Due to the requirement that the library be position independent (i.e. sharable), loading is performed by an indirect load. For this indirect load, the address of the pointer to the constant is computed first, the pointer to the constant is then loaded, then the constant is loaded. For a processor, such as Intel's Itanium™, this sequence takes a minimum of 13 cycles. This sequence can take longer than 13 cycles if the pointer and constants are not available in cache memory.

On some processors, such as Intel's Itanium™, there is no method to directly load a double-extended constant without using memory instructions. There is a way, however, to directly load the significand of a floating-point constant by first forming a 64-bit significand in an integer register and then using an instruction (e.g., setf.sig on Intel's Itanium™) to put the significand into the floating-point register. Such an instruction sets the exponent to 2^{63}. On a processor, such as the Intel Itanium™ processor, this sequence takes 10 cycles. In one embodiment of the invention, three cycles of latency can be saved by using a constant S, having the correct significand, but a modified exponent.

Figure 4 illustrates an embodiment of the invention used to generalize selection of a constant S in determining n_i, n_f, and r. In process 400, block 410 computes the result of x*A'+S' (where S' is a scaled version of S, discussed further below). Block 420, using the result from block 410, multiplies the result from block 410 by T (T is a factor, where $T = 2^{-(b-1-j)}$) and then subtracts S. Block 430 extracts the significand bits from the result from block 410, thus creating an integer value. Block 440 computes r by computing $x - n_f*B$. Block 450 extracts the low-order bits from the result of block 430. At block 460, n_i is available to be transmitted to an ALU or stored in memory. In block 470, r is available to be transmitted to an ALU or stored in memory. In process 400, $A = 2^j*F$, where F is the significand of the form 1.xxxxxxxx, $1.0 \leq |F| < 2.0$. Also, $A' = 2^{b-1}*F$.

Table III illustrates pseudo-code steps for process 400 illustrated in **Figure 4.**

Table III

Falu op 1:	w_plus_S_rshifted = x * A' + S'	(1)
Falu op 2:	n_f = w_plus_S_rshifted * T – S	(6)
Ialu op 1:	ni_plus_S = extract_significand_bits (w_plus_S_shifted)	(9)
Falu op 3:	r = x - n_f * B	(11)
Ialu op 2:	n_i = extract_low_order_bits(ni_plus_S)	(11)
	n_i available	(12)
	r available	(16)

In one embodiment of the invention, for the shift to performed properly, a scaled version of S is needed, S', in Falu op 1, where $S' = S * 2^{b-1-j}$. To get n_f in Falu op 2, w_plus_S_rshifted is scaled back by a factor T, where $T = 2^{-(b-1-j)}$. In this embodiment of the invention, four constants are generated, A', S', S, and T. In one embodiment of the invention, these four constants are determined in parallel.

Figure 5 illustrates process 500, which is a typical process for loading constants and calculating coefficients for decomposition of floating-point numbers into their integer and fractional parts. On a typical processor, such as Intel's Itanium™, the entire sequence from loading constants through the computation of r, takes 36 cycles. Process 500 begins with block 510 which computes the address of a pointer to A and B. Block 520 loads the address of the pointer to A and B. Block 530 loads A and B. Block 540 computes the equation w = x*A. Block 550 converts the result from block 540 (w) to an unnormalized integer. Block 560 computes n_f by converting the result of block 550 to a normalized whole number. Block 570 computes n_i by converting the result of block 550 to an integer. In block 580, n_i is available to be transmitted to an ALU or stored in memory. In block 590, r is computed by the equation x - n_f * B. In block 595, r is available to be transmitted to an ALU or stored in memory.

Table IV illustrates process 500 in pseudo-code. The numbers on the right hand side of **Table IV** represent typical cycles on a processor such as Intel's Itanium™.

Table IV

Ialu op 1:	Compute address of pointer to A,B	(1)
Ialu op 2:	Load address of pointer to A,B	(2)
Ialu op 3:	Load A,B	(5)
Falu op 1:	w = x * A	(14)

Falu op 2:	w_rshifted = convert_to_unnormalized_rounded_int(w)	(19)
Falu op 3:	n_f = convert_to_normalized_whole_number(w_rshifted)	(26)
Ialu op 4;	n_i = convert_to_integer(w_rshifted)	(27)
	n_i available	(29)
Falu op 4:	$r = x - n_f * B$	(31)
	r available	(36)

Figure 6A-B illustrates an embodiment of the invention for loading of constants and performing decomposition of floating-point numbers into their integer and fractional parts. Process 600 begins with block 605 which forms a bit pattern of S' in an integer register. Block 610 forms a bit pattern of the significand of A in an integer register. Block 615 creates S' in a floating-point register. Block 620 creates A' in a floating-point register. Block 625 forms a bit pattern of S in an integer register. Block 630 forms a bit pattern of T in an integer register. Block 635 computes the address of a pointer to B. Block 640 creates S in a floating-point register. Block 645 creates T in a floating-point register. Block 650 loads the address of a pointer to B. Block 655 loads B. Block 660 computes $x * A' + S'$. In block 665, the result from block 660 is multiplied by T and then the value for S is subtracted. The result from block 665 represents n_f. In block 670, the significand bits are extracted from the result from block 660, thus creating an integer value. In block 675, r is computed by the equation $x - n_f * B$. In block 680, the result from block 670 is used to extract the low order of bits. The result of block 680 is n_i. In block 685, n_i is available to be transmitted to an ALU or stored in memory. In block 690 r is available to be transmitted to an ALU or stored in memory.

Table V illustrates process 600 (see **Figure 6A-B**) in pseudo-code format. Note that the numbers on the right of **Table V** enclosed in parentheses represent cycles for a processor, such as Intel's Itanium™. In one embodiment of the invention, process 300 and process 600 are loaded into mathematical libraries used by various compilers. In another embodiment of the invention, the same processes loaded into a mathematical library can be used for processing functions, such as scalar double precision tangent, sine, cosine, exponential functions, hyperbolic cosine, hyperbolic sine, hyperbolic tangent, etc. to reduce the number of cycles necessary to complete operations as compared to prior art. It should be noted that other embodiments of the

invention can be used for processing functions such as scalar single precision, vector double precision, and vector single precision.

Table V

Ialu op 1:	Form bit pattern of S' in integer reg (movl)	(1)
Ialu op 2:	Form bit pattern significand of A in integer reg(movl)	(1)
Ialu op 3:	Create S' in fp reg (setf.d)	(2)
Ialu op 4:	Create A' in fp reg (setf.sig)	(2)
Ialu op 5:	Form bit pattern of S in integer reg (movl)	(2)
Ialu op 6:	Form bit pattern of T in integer reg (movl)	(2)
Ialu op 7:	Compute address of pointer to B	(3)
Ialu op 8:	Create S in fp reg (setf.d)	(4)
Ialu op 9:	Create T in fp reg (setf.d)	(4)
Ialu op 10:	Load address of pointer to B	(5)
Ialu op 11:	Load B	(8)
Falu op 1:	w_plus_S_rshifted = x * A' + S'	(11)
Falu op 2:	n_f = w_plus_S_rshifted * T – S	(16)
Ialu op 12:	ni_plus_S = extract_significand_bits (w_plus_S_rshifted)	(19)
Falu op 3:	r = x - n_f * B	(21)
Ialu op 13:	n_i = extract_low_order_bits(ni_plus_S)	(21)
	n_i available	(22)
	r available	(26)

Figure 7 illustrates an embodiment of the invention having computational component 710. Circuit 700 also comprises microprocessor 720, cache 730, memory 740, disk storage 750, pre-fetch queue 755, decode/assignment/predictor 760, integer pipeline A 770, integer pipeline B 775, floating-point pipeline A 780, ALU 781-782, floating point ALU 783, integer register sets 785-786, floating point register set 787, and data bus 790. In one embodiment of the invention, computational component 710 incorporates process 300, 400 or 600 illustrated in **Figures 3, 4,** and **6A-B,** respectively.

The above embodiments of the invention can be used whenever integer and fractional components of a floating-point number are necessary to perform argument reduction portions of scalar and vector double precision functions, scalar and vector single precision functions, various mathematical functions, and preprocessing before computing mathematical functions. By using the above discussed embodiments of the invention, computational latency is reduced without compromising precision.

The above embodiments can also be stored on a device or machine-readable medium and be read by a machine to perform instructions.

The machine-readable medium includes any mechanism that provides (i.e., stores and/or transmits) information in a form readable by a machine (e.g., a computer). For example, a machine-readable medium includes read only memory (ROM); random access memory (RAM); magnetic disk storage media; optical storage media; flash memory devices; electrical, optical, acoustical or other form of propagated signals (e.g., carrier waves, infrared signals, digital signals, etc.). The device or machine-readable medium may include a solid state memory device and/or a rotating magnetic or optical disk. The device or machine-readable medium may be distributed when partitions of instructions have been separated into different machines, such as across an interconnection of computers.

While certain exemplary embodiments have been described and shown in the accompanying drawings, it is to be understood that such embodiments are merely illustrative of and not restrictive on the broad invention, and that this invention not be limited to the specific constructions and arrangements shown and described, since various other modifications may occur to those ordinarily skilled in the art.

CLAIMS:

What is claimed is:

1. A method comprising:

decomposing a first value into a plurality of parts, said decomposing comprises:

shifting a rounded integer portion of the first value to generate a second value;

generating a third value from the second value;

extracting a plurality of significand bits from the second value to generate a fourth value;

generating a fifth value from the third value; and

extracting a portion of bits from the fourth value to generate an integer component,

wherein the third value, the fifth value, and the integer component are one of stored in a memory and transmitted to an arithmetic logical unit (ALU).

2. The method of claim 1, further comprising:

generating a first constant.

3. The method of claim 2, wherein the first value, the first constant and the second value are floating-point formats.

4. The method of claim 1, wherein the rounded integer portion is shifted into a rightmost bits portion of a significand of the first value.

5. The method of claim 1, wherein shifting the rounded integer portion comprises selecting the first constant so that adding the first constant to the first value shifts the rounded integer portion.

6. A method comprising:

decomposing a first value into a plurality of parts, decomposing comprises:

generating a first constant;

scaling the first constant by a factor to result in a second constant;

generating a second value by shifting a rounded integer portion of the first value;

extracting a plurality of significand bits from the second value to generate an integer value; and

extracting a portion of bits from the integer value to generate an integer component.

7. The method of claim 6, wherein the rounded integer portion is shifted into a rightmost bits portion of a significand of the first value.

8. The method of claim 6, wherein the first constant, the second constant, the first value, and the second value are floating-point quantities.

9. The method of claim 6, wherein shifting the rounded integer portion comprises adding the second constant to the first value.

10. A method comprising:

generating a first constant;

scaling the first constant to generate a second constant;

forming a bit pattern for the first constant, the second constant, a third constant, and a fourth constant in separate integer registers;

determining an address for a pointer to a fifth constant;

loading the address for the pointer to the fifth constant in a memory;

generating a second value;

extracting a plurality of significand bits from the second value to generate an integer value; and

extracting a portion of bits from the integer value to generate an integer component;

11. The method of claim 10, wherein the rounded integer portion is shifted into a rightmost bits portion of a significand of the first value.

12. The method of claim 10, further comprising:

creating a representation of the first constant in a first floating-point register;

creating a representation of the second constant in a second floating-point register;

creating a representation of a scaled version of the third constant in a third floating point register; and

creating a representation of the fourth constant in a fourth floating point register.

13. The method of claim 10, wherein the first constant, the second constant, the third constant , the fourth constant, the first value, and the second value are floating-point formats.

14. An apparatus comprising a machine-readable medium containing instructions which, when executed by a machine, cause the machine to perform operations comprising:

decomposing a first value into a plurality of parts, the instructions that cause the machine to perform decomposing operations further includes operations including:

generating a first constant;

shifting a rounded integer portion of the first value to generate a second value;

generating a third value from the second value;

extracting a plurality of significand bits from the second value to generate a fourth value;

generating a fifth value from the third value; and

extracting a portion of bits from the fourth value to generate an integer component,

wherein the third value, the fifth value, and the integer component are one of stored in a memory and transmitted to an arithmetic logical unit (ALU).

15. The apparatus of claim 14, wherein the rounded integer portion is shifted into a rightmost bits portion of a significand of the first value.

16. The apparatus of claim 14, wherein the first value, the first constant, the second constant, and the second value are floating-point formats.

17. An apparatus comprising a machine-readable medium containing instructions which, when executed by a machine, cause the machine to perform operations comprising:

decomposing a first value into a plurality of parts, the instructions that cause the machine to perform decomposing operations further includes operations including:

generating a first constant;

scaling the first constant by a factor to result in a second constant;

generating a second value by shifting a rounded integer portion of the first value;

extracting a plurality of significand bits from the second value to generate an integer value; and

extracting a portion of bits from the integer value to generate an integer component.

18. The apparatus of claim 16, wherein the rounded integer portion is shifted into a rightmost bits portion of a significand of the first value.

19. The apparatus of claim 16, wherein the first constant, the second constant, the first value, and the second value, are floating-point formats.

20. An apparatus comprising a machine-readable medium containing instructions which, when executed by a machine, cause the machine to perform operations comprising:

generating a first constant;

generating a second constant;

forming a bit pattern for the first constant, the second constant, a third constant, and a fourth constant in separate integer registers;

determining an address for a pointer to a fifth constant;

loading the address for the pointer to the fifth constant in a memory;

generating a second value;

generating a third value;

extracting a plurality of significand bits from the second value to generate an integer value;

extracting a portion of bits from the integer value resulting in an integer component;

generating a fourth value; and

storing the fourth value, the integer component, and the third value in a memory.

21. The apparatus of claim 20, wherein the rounded integer portion is shifted into a rightmost bits portion of a significand of the first value.

22. The apparatus of claim 20, further containing instructions which, when executed by a machine, cause the machine to perform operations including:

creating a representation of the first constant in a first floating-point register;

creating a representation of the second constant in a second floating-point register;

creating a representation of a scaled version of the third constant in a third floating point register; and

creating a representation of the fourth constant in a fourth floating point register.

23. The apparatus of claim 20, wherein the first constant, the second constant, the third constant, the fourth constant, the first value, the second value, and the third value are floating-point formats.

24. An apparatus comprising:

a processor, the processor having a computational component;

a bus coupled to the processor;

a memory coupled to the processor;

a plurality of arithmetic logical units (ALUs) coupled to the processor; and

a plurality of register sets coupled to the plurality of ALUs,

wherein the computational component generates a first constant, generates a second value, generates a third value, extracts a plurality of significand bits from the second value to generate an integer value, generates a fourth value, extracts a portion of bits from the integer value to generate an integer

component, and stores the fourth value, the integer component, and the third value in the memory.

25. The apparatus of claim 24, wherein the rounded integer portion is shifted into a rightmost bits portion of a significand of the first value.

26. The apparatus of claim 24, wherein the first value, the first constant, the second constant, and the second value are floating-point formats.

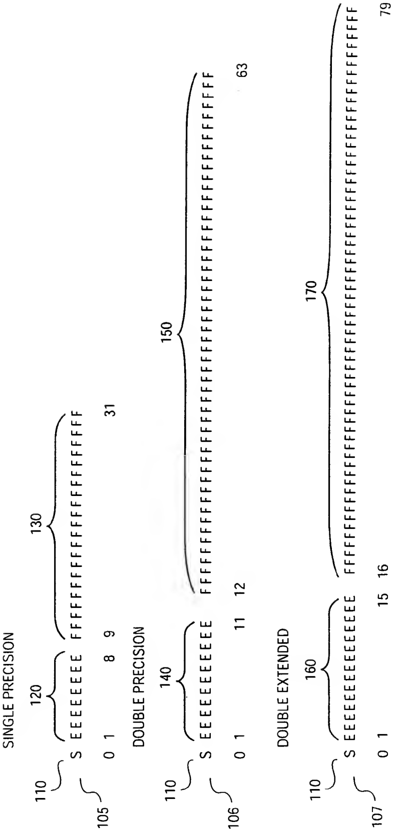

SINGLE PRECISION
DOUBLE PRECISION
DOUBLE EXTENDED
105
106
107
110
120
130
140
150
160
170
0 1 8 9 31
0 1 11 12 63
0 1 15 16 79

FIG. 1

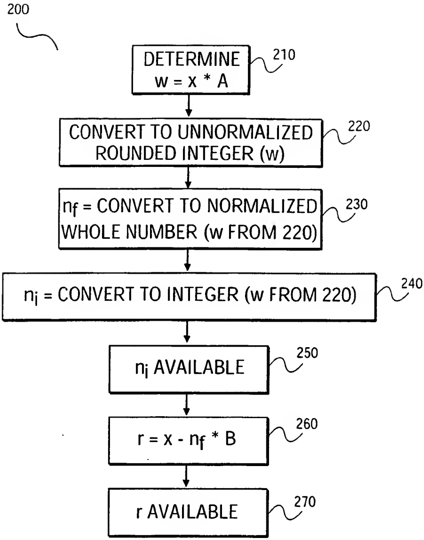

200
DETERMINE w = x * A
210
CONVERT TO UNNORMALIZED ROUNDED INTEGER (w)
220
n_f = CONVERT TO NORMALIZED WHOLE NUMBER (w FROM 220)
230
n_i = CONVERT TO INTEGER (w FROM 220)
240
n_i AVAILABLE
250
r = x - n_f * B
260
r AVAILABLE
270

FIG. 2

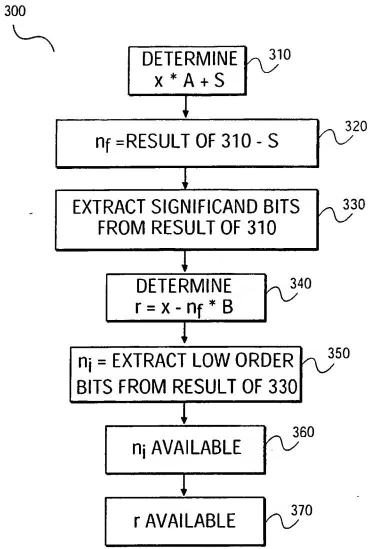

300
DETERMINE
x * A + S
310
n_f =RESULT OF 310 - S
320
EXTRACT SIGNIFICAND BITS
FROM RESULT OF 310
330
DETERMINE
$r = x - n_f * B$
340
n_i = EXTRACT LOW ORDER
BITS FROM RESULT OF 330
350
n_i AVAILABLE
360
r AVAILABLE
370

FIG. 3

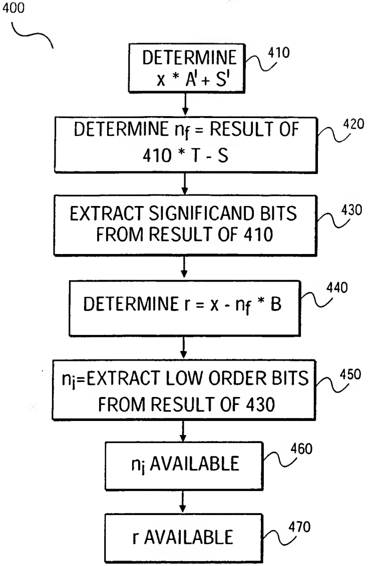

400
DETERMINE x * A' + S'
410
DETERMINE nf = RESULT OF 410 * T - S
420
EXTRACT SIGNIFICAND BITS FROM RESULT OF 410
430
DETERMINE r = x - nf * B
440
ni=EXTRACT LOW ORDER BITS FROM RESULT OF 430
450
ni AVAILABLE
460
r AVAILABLE
470

FIG. 4

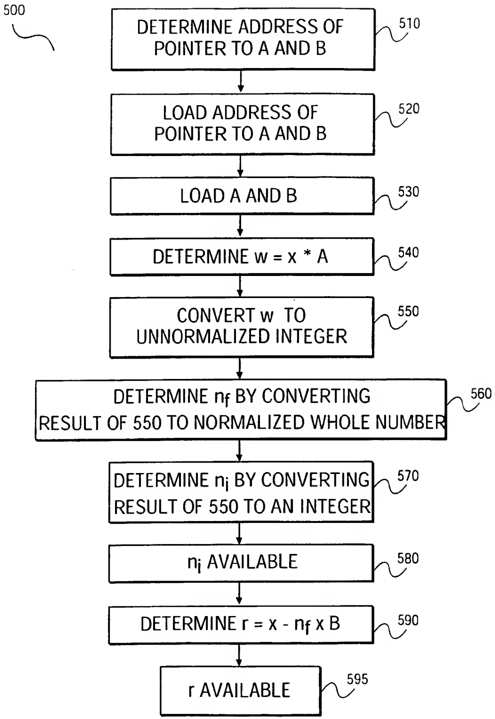

500
DETERMINE ADDRESS OF POINTER TO A AND B
510
LOAD ADDRESS OF POINTER TO A AND B
520
LOAD A AND B
530
DETERMINE w = x * A
540
CONVERT w TO UNNORMALIZED INTEGER
550
DETERMINE nf BY CONVERTING RESULT OF 550 TO NORMALIZED WHOLE NUMBER
560
DETERMINE ni BY CONVERTING RESULT OF 550 TO AN INTEGER
570
ni AVAILABLE
580
DETERMINE r = x - nf x B
590
r AVAILABLE
595

FIG. 5

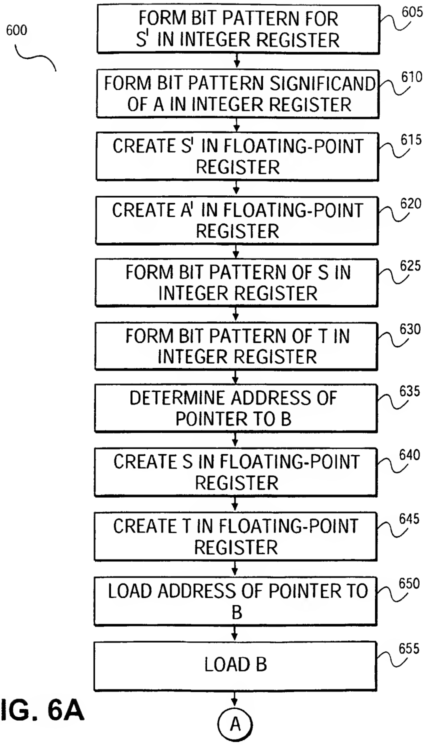
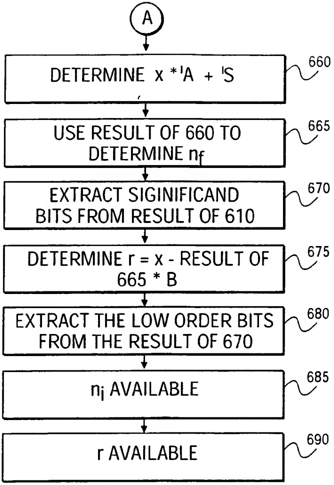

600
FORM BIT PATTERN FOR S' IN INTEGER REGISTER
605
FORM BIT PATTERN SIGNIFICAND OF A IN INTEGER REGISTER
610
CREATE S' IN FLOATING-POINT REGISTER
615
CREATE A' IN FLOATING-POINT REGISTER
620
FORM BIT PATTERN OF S IN INTEGER REGISTER
625
FORM BIT PATTERN OF T IN INTEGER REGISTER
630
DETERMINE ADDRESS OF POINTER TO B
635
CREATE S IN FLOATING-POINT REGISTER
640
CREATE T IN FLOATING-POINT REGISTER
645
LOAD ADDRESS OF POINTER TO B
650
LOAD B
655
A

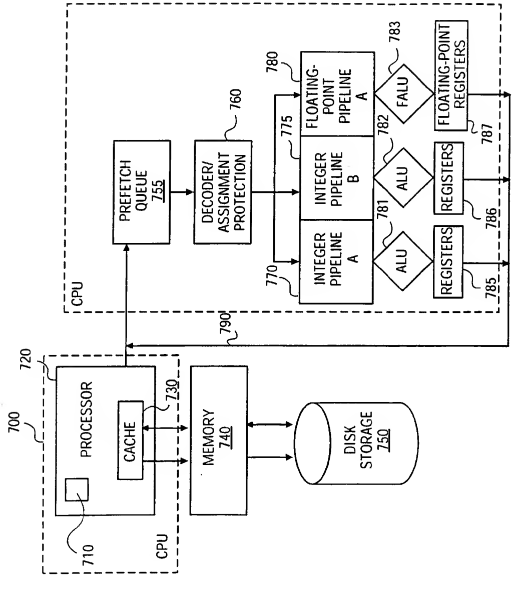
FIG. 6A



A
DETERMINE x * 'A + 'S
660
USE RESULT OF 660 TO DETERMINE n_f
665
EXTRACT SIGINIFICAND BITS FROM RESULT OF 610
670
DETERMINE r = x - RESULT OF 665 * B
675
EXTRACT THE LOW ORDER BITS FROM THE RESULT OF 670
680
n_i AVAILABLE
685
r AVAILABLE
690

FIG. 6B

FIG. 7


700
720
PROCESSOR
CACHE
730
710
CPU
MEMORY 740
DISK STORAGE 750
790
CPU
PREFETCH QUEUE 755
DECODER/ ASSIGNMENT PROTECTION
760
770
INTEGER PIPELINE A
775
INTEGER PIPELINE B
780
FLOATING-POINT PIPELINE A
781
ALU
782
ALU
783
FALU
REGISTERS
785
REGISTERS
786
FLOATING-POINT REGISTERS
787

INTERNATIONAL SEARCH REPORT

International application No.
PCT/RU 01/00286

A. CLASSIFICATION OF SUBJECT MATTER

G06F 7/38

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F 7/00, 7/38, 7/40, 7/42, 7/44, 7/46, 7/49, 7/50, 7/52, 7/54, 7/60

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched:

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No
A	US 6205460 B1 (SUN MICROSYSTEMS, INC.) Mar. 20, 2001	1-26
A	US 6038582 A (HITACHI, LTD.) Mar. 14, 2000	1-26
A	SU 1546964 A1 (SPETSIALNOE PROEKTNO-KONSTRUKTORSKOE BYURO "DISKRET" ODESSKOGO POLITEKHNICHESKOGO INSTITUTA) 28.02.1990	1-26
A	SU 1259248 A1 (V.M. BORISOVA et al) 23.09.1986	1-26
A	SU 1513443 A1 (INSTITUT KIBERNETIKI IM. B.M. GLUSHKOVA) 07.10.1989	1-26
A	US 5940311 A (TEXAS INSTRUMENTS INCORPORATED) Aug. 17, 1999	1-26
A	SU 1361542 A1 (INSTITUT PRIKLADNOY MATEMATIKI im. M.V. KELDYSHA) 23.12.1987	1-26

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document with may throw doubts on priori claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the documentis taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search 18 March 2002 (18.03.2002)	Date of mailing of the international search report 28 March 2002 (28.03.2002)
Name and mailing address of the ISA/RU FIPS Russia, 123995, Moskva, G-59, GSP-5 Berezhkovskaya nab., 30-1 Facsimile No.	Authorized officer O. Krysanova Telephone No. (095)240-25-91